Lehrstuhl für Praktische Informatik IV
Holger Füßler
A5, 6, Raum B 219
68131 Mannheim
Telefon: (0621) 181–2605
Email: fuessler@informatik.uni-mannheim.de

Robert Schiele
B6, 29, Raum C0.04
68131 Mannheim
Telefon: (0621) 181–2214
Email: rschiele@uni-mannheim.de

UNIVERSITÄT MANNHEIM

Praktische Informatik I — 7. Übungsblatt
Wintersemester 2005/2006 — Abgabe: 14. Dezember 2005

Allgemeine Bearbeitungshinweise

Verwenden Sie für die Programmieraufgaben die in den Aufgaben angegebenen Klassennamen und geben Sie den Klassen-Quellcode sowohl auf Papier als auch per e-mail bei Ihrem Tutor ab. Falls nicht anders gefordert, lagern Sie sinnvoll Funktionalität in Methoden aus. Verwenden Sie Javadoc-Kommentare zur Dokumentation.

Aufgabe 1 — $\Sigma = 12$ Punkte

Aufgabe 1 a) — 8 Punkte

Implementieren Sie den aus der Vorlesung bekannten rekursiven Algorithmus zur Lösung des Problems der Türme von Hanoi. Verwenden Sie hierzu die Klasse `Hanoi`, welche Sie unter `http://tinyurl.com/azdcx` herunterladen können, und erweitern Sie diese Klasse in Ihrer eigenen Klasse mit dem Namen `MyHanoi`.
Die Klasse Hanoi hat die folgende Signatur:

```
1 public class Hanoi {
    public Hanoi(int disks, boolean show);
    public void move(int source, int destination) throws Exception;
    public int disksOnStack(int stack);
  }
```

Der Konstruktor erzeugt ein Objekt dieser Klasse mit drei Stapeln (1–3) und der übergebenen Anzahl Scheiben, welche sich initial auf dem Stapel 1 befinden. Dem Konstruktor wird außerdem ein boolescher Wert übergeben, der angibt, ob automatisch nach jedem Schritt eine Grafik des aktuellen Zustands ausgedruckt werden soll.
Die Methode `move` bewegt eine Scheibe vom Stapel `source` auf den Stapel `destination`. Sind die Parameter ungültig, so wirft die Methode eine Exception.
Die Methode `disksOnStack` gibt zurück, wie viele Scheiben sich momentan auf dem übergebenen Stapel befinden.
Implementieren Sie in Ihrer Klasse `MyHanoi`, welche von `Hanoi` abgeleitet sein soll, eine Methode mit der Signatur `void moveall(int source, int destination)`, welche alle Scheiben vom Stapel `source` auf den Stapel `destination` verschiebt. Sofern notwendig oder sinnvoll, implementieren Sie weitere hierzu benötigte Methoden.

Aufgabe 1 b) 4 Punkte

Implementieren Sie eine `main`-Methode in derselben Klasse, welche testet, ob Ihre Implementation des Algorithmus funktioniert. Diese Methode soll außerdem die Zeit messen, die zur Ausführung des Algorithmusses benötigt wird. Benutzen Sie hierzu die Methode `java.lang.System.nanoTime`. Messen Sie mit Ihrem Rechner die Laufzeit für 5, 10, 15, 20, 25 und 30 Scheiben. Schalten Sie hierzu die grafische Ausgabe ab. Was lässt sich über das Laufzeitverhalten in Abhängigkeit zur Anzahl der Scheiben sagen?

Aufgabe 2 — $\Sigma = 9$ Punkte

Eine berühmte Zahlenfolge sind die sogenannten Fibonacci-Zahlen. Sie sind folgendermaßen definiert:

$$\begin{aligned} F_0 &= 0; \\ F_1 &= 1; \\ F_N &= F_{n-1} + F_{n-2}. \qquad\qquad \forall n > 1. \end{aligned}$$

Aufgabe 2 a) 4 Punkte

Schreiben Sie ein Programm (Klassenname `Fibonacci`), das jeweils eine iterative und eine rekursive Methode für die Berechnung des n-ten Reihenelements der Fibonacci-Reihe bietet.

Aufgabe 2 b) 3 Punkte

Testen Sie Ihre Methoden für $n = 1, 5, 10, 20, 50$. Geben Sie für jeden der Werte die Laufzeit der Methoden aus (siehe *Hanoi* Aufgabe).

Aufgabe 2 c) 2 Punkte

Wie lassen sich die Laufzeitunterschiede zwischen iterativer und rekursiver Methode erklären?

Aufgabe 3 — $\Sigma = 8$ Punkte

Aufgabe 3 a) 6 Punkte

In der Vorlesung wurde eine Java-Implementation für eine einfach verkettete Liste vorgestellt. Implementieren Sie die selben Methoden für eine *doppelt* verkettete Liste mit dem Namen `DoubleLinkedList`, das heißt implementieren Sie das Interface `List`, welches Sie unter `http://tinyurl.com/avsy7` finden. Eine doppelt verkettete Liste unterscheidet sich von einer einfach verketteten Liste dadurch, dass jedes Listenelement nicht nur einen Zeiger auf das nachfolgende Element hat, sondern auch auf das vorhergehende.

Aufgabe 3 b) 2 Punkte

Welche der für die doppelt verkettete Liste implementierten Methoden hat ihr Laufzeitverhalten gegenüber der Variante für die einfach verkettete Liste *wesentlich* geändert? Ist das Laufzeitverhalten an diesen Stellen jeweils besser oder schlechter geworden? Was ist die Ursache hierfür?

Aufgabe 4 6 Punkte

Textaufgabe: Zeigen Sie mittels vollständiger Induktion, dass die folgende Java-Methode für alle positiven Ganzzahlen deren Quadratzahlen korrekt berechnet:

```
public static int sqr(int x) {
    return (x == 0) ? 0 : sqr(x − 1) + 2 ∗ x − 1;
}
```